First-Class Distributed Session Types

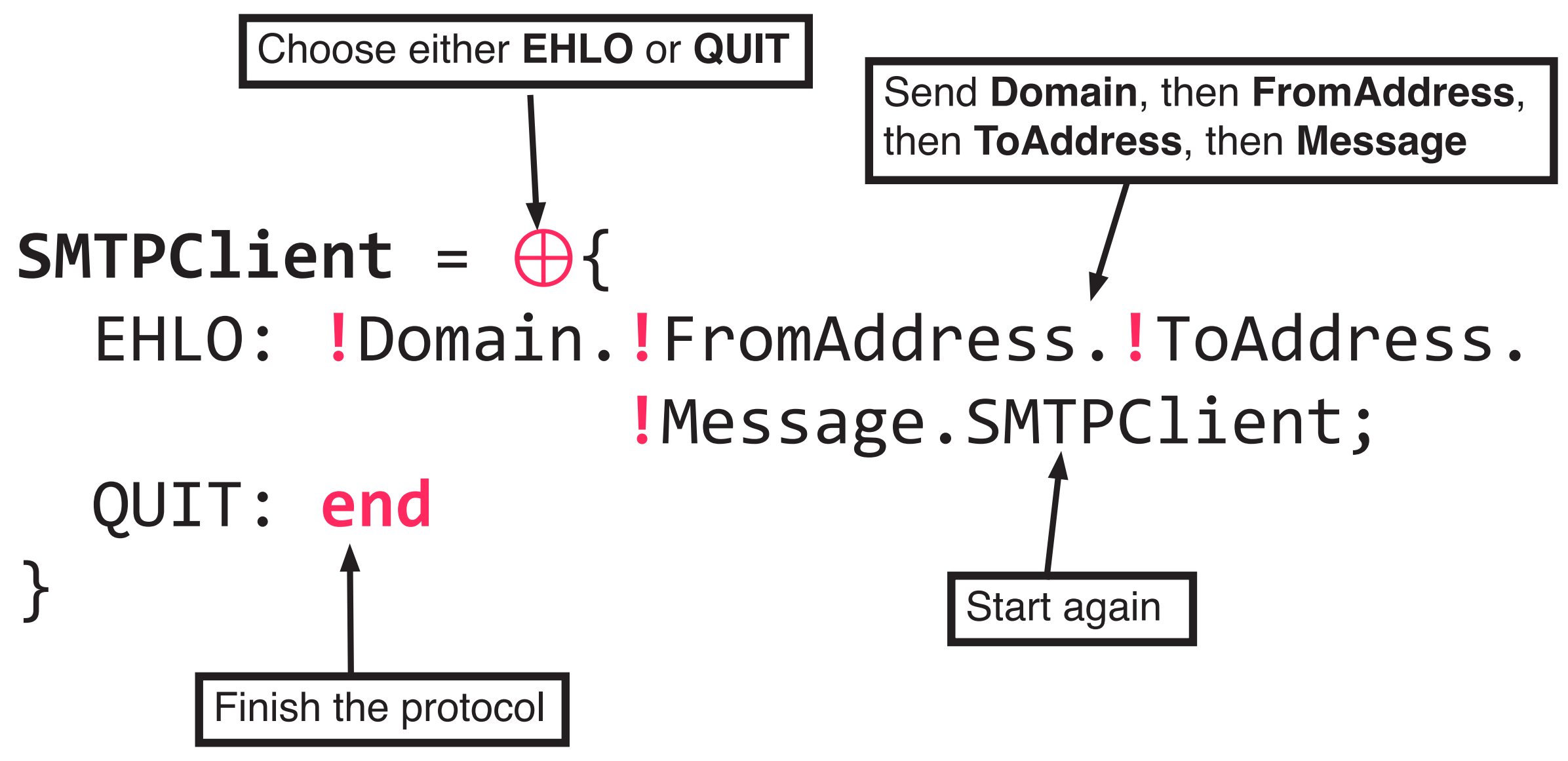
Simon Fowler
simon.fowler@ed.ac.uk
Joint work with Sam Lindley and J. Garrett Morris

THE UNIVERSITY of EDINBURGH
EPSRC Engineering and Physical Sciences Research Council
EPSRC Centre for Doctoral Training in Pervasive Parallelism

Session Types: Types for Protocols

Data types classify data.

Session types describe **protocols** as types.



Choose either **EHLO** or **QUIT**

Send **Domain**, then **FromAddress**, then **ToAddress**, then **Message**

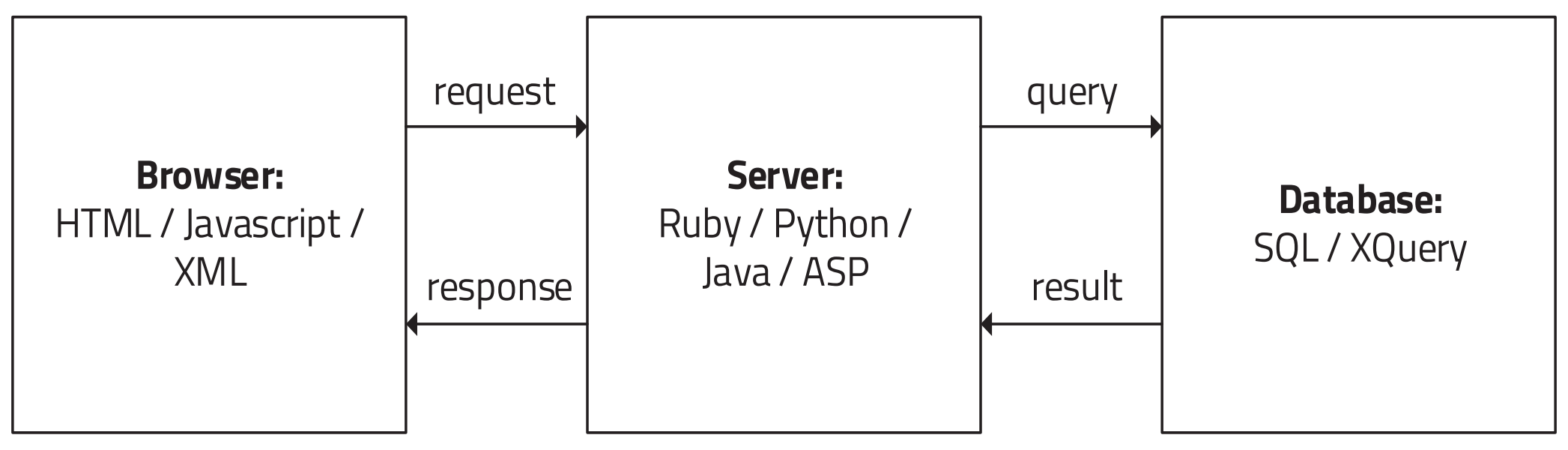
```
SMTPClient = ⊕{
  EHLO: !Domain.!FromAddress.!ToAddress.
               !Message.SMTPClient;
  QUIT: end
}
```

Start again

Finish the protocol

Links: Web Programming without Tiers

Links: a **functional web language** unifying client, server and database code.

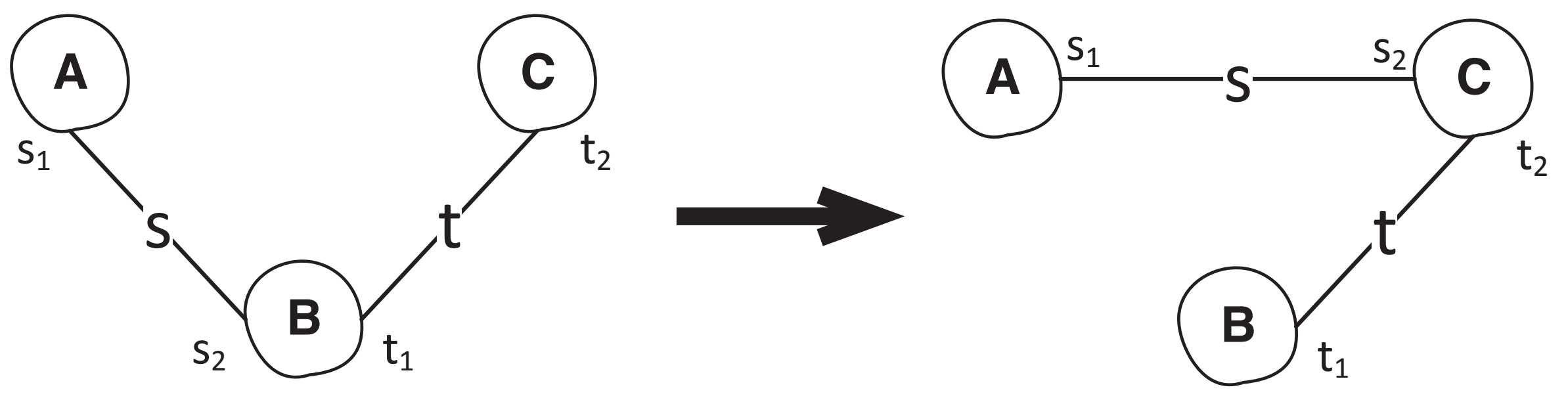


Links extended with session types [2], making use of a **linear type system**.

This work: **Distributed** Session Links.

- Session abstractions in the **web setting**
- Web-based **Distributed Delegation**
- Theory and practice of **session exception handling**

Distributed Delegation



Delegation: sending channels over channels.

In distributed setting, issue: "lost messages".

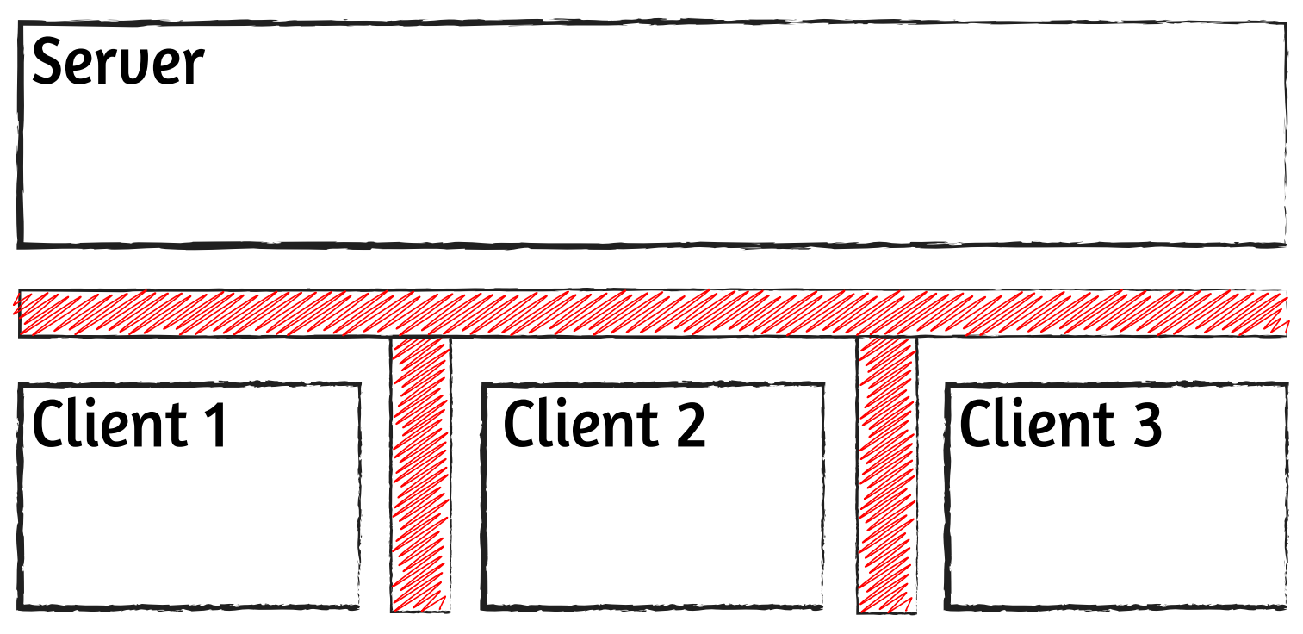
- A wants to send 5 along s_1, B wants to send s_2 along t_1.

No happens-before relation! 5 may be sent to B, not C.

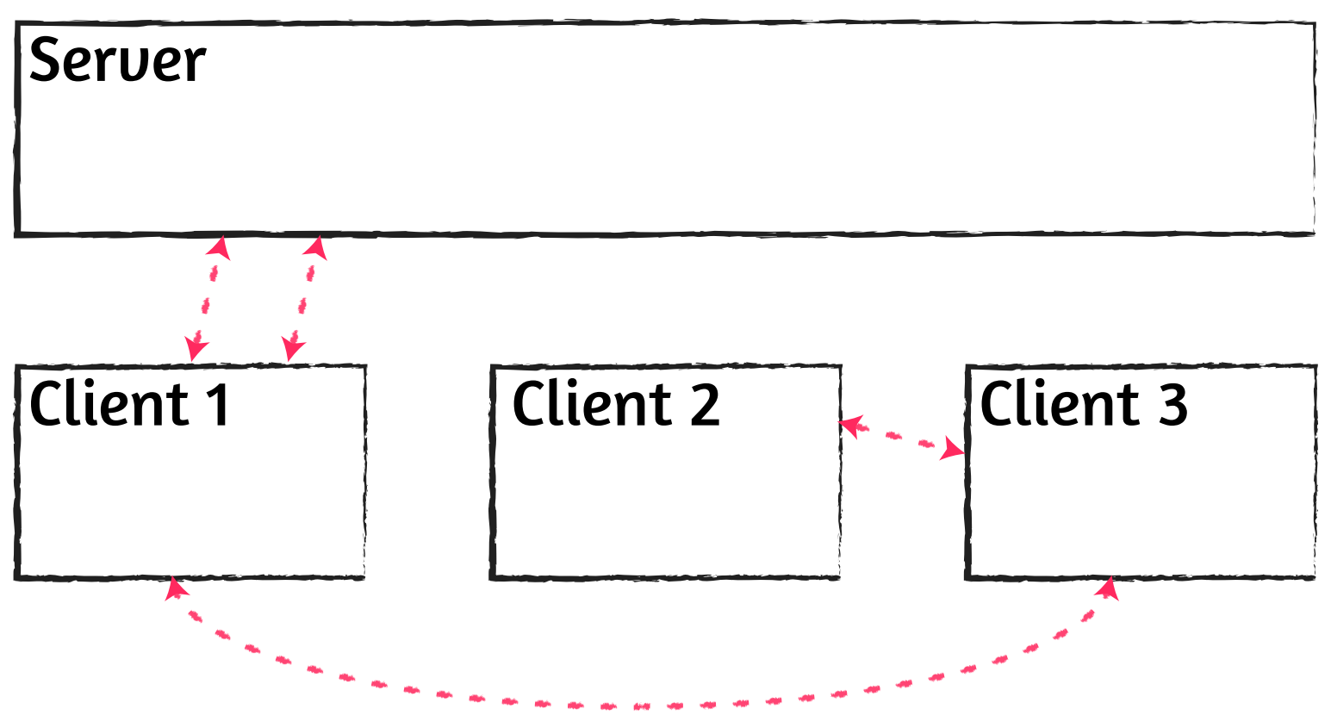
- **Inspect** sent messages, send delegated buffers
- **Update** endpoint locations on server
- **Retrieve** lost messages, forward to recipient
- **Final buffer at recipient:** initial buffer + lost messages + messages received after lost messages.

Must ensure carrier channels aren't delegated!

Breaking the Barrier: From Multithreaded to Distributed



Concurrency on server and clients. **Limitation**: no communication across boundaries!



Distributed Session Links: breaks barriers. **Websockets** allow full-duplex communication between clients and server.

- Automatic message serialisation and deserialisation
- Distribution-aware concurrency runtimes
- Distributed algorithm for channel mobility

"But what happens if a user just closes the browser?": Affine Sessions with Exceptional Syntax

```
sig recvAndAdd :
  (?Int.end, ?Int.end) ~> Int
fun recvAndAdd(s, t){
  try {
    let (x, s) = receive s in
    let (y, t) = receive t in
    (x, y)
  } as (x, y) in {
    x + y
  } otherwise {
    (-1)
  }
}
```

Affine sessions: a user can close their browser!

Exception construct with **explicit success continuation** [1]. On communication error:

- **Inspect** free variables, **cancel** affected channels
- Proceed to "otherwise" block if exception handled
- Halt thread otherwise

Adapts **Affine Sessions** [3] to asynchronous concurrent λ-calculus GV.

Formalism

Example Reduction Rules

Receiving a Message

$$(\nu a)(F[\mathsf{receive}\,a] \parallel a(V' \cdot \overrightarrow{V}) \leftrightsquigarrow b(Q)) \longrightarrow_{\mathsf{C}} (\nu a)(F[(V', a)] \parallel a(\overrightarrow{V}) \leftrightsquigarrow b(Q))$$

Successful receive: Process receives value *V'* from buffer *a*.

Receiving a Message: Exception Raised

$$(\nu a)(M \parallel a(\varepsilon) \leftrightsquigarrow b(\lightning)) \quad \longrightarrow_{\mathsf{C}} \quad (\nu a)(F[N'] \parallel \lightning a \parallel \lightning c_1 \parallel \ldots \parallel \lightning c_n \parallel a(\varepsilon) \leftrightsquigarrow b(\lightning))$$
$$\text{where } M = F[\mathsf{try}\, E[\mathsf{receive}\, a] \,\mathsf{as}\, x \,\mathsf{in}\, N \,\mathsf{otherwise}\, N']$$
$$\mathsf{fvs}(E[\mathsf{receive}\, a]) = \{c_i\}_{i \in 1..n} \cup \{a\}$$

Unsuccessful handled receive: Process tries to receive from empty buffer where peer endpoint is cancelled. Cancel free variables in context; evaluate `otherwise` term.

Channel Cancellation

$$(\nu a)(\lightning a \parallel a(\overrightarrow{V}) \leftrightsquigarrow b(Q)) \quad \longrightarrow_{\mathsf{C}} \quad (\nu a)(\lightning a \parallel \lightning c_1 \parallel \ldots \parallel \lightning c_n \parallel a(\lightning) \leftrightsquigarrow b(Q))$$
$$\text{where } \mathsf{fvs}(\overrightarrow{V}) = \{c_i\}_{i \in 1..n}$$

Channel Cancellation: Cancel all names contained within a buffer.

Preservation (Configuration Reduction)

Reduction preserves typeability of configurations C. Compatibility relation $\asymp$ on typing contexts.

Theorem: Assume Γ only contains channel names. If $\Gamma;\Delta \vdash \mathcal{C}$ and $\mathcal{C} \rightarrow_{\mathsf{C}} \mathcal{C}'$, then there exist Γ', Δ' such that $\Gamma, \Delta \asymp \Gamma', \Delta'$ and $\Gamma', \Delta' \vdash \mathcal{C}'$.

Deadlock-Freedom and Progress

Calculus inherits deadlock-freedom from logical roots of GV.

Theorem: Suppose $\Gamma;\Delta \vdash \mathcal{C}$ and $\mathcal{C} \not\rightarrow_{\mathsf{C}}$. If the main thread has been cancelled, then $\mathcal{C} \equiv \mathbf{halt}$. Otherwise, if the value returned by C contains no channels, then $\mathcal{C} \equiv V$.

Future Work

Current status: full communication between different concurrency runtimes, formalism and metatheory for exceptions.

Exception Implementation: Implement exception handling mechanism in Links: CEK extension for interpreter; CPS extension for client.

Multiple Servers: Inspired by Hop.js services [4], allow multiple Links servers.

References:
[1] Benton, N. and Kennedy, A., 2001. Exceptional syntax. Journal of Functional Programming, 11(4), pp.395-410.
[2] Lindley, S. and Morris, J.G., 2017. Lightweight Functional Session Types. Behavioural Types: from Theory to Tools, p.265.
[3] Mostrous, D. and Vasconcelos, V.T., 2014. Affine sessions. In International Conference on Coordination Languages and Models (pp. 115-130). Springer, Berlin, Heidelberg.
[4] Serrano, M. and Prunet, V., 2016. A glimpse of Hopjs. In International Conference on Functional Programming (ICFP) (p. 12).